Using Analogical Reasoning to Promote Creativity in Software Reuse

Paulo Gomes, Francisco C. Pereira, Carlos Bento and José Luís Ferreira

Centro de Informática e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informática, Polo II, Universidade de Coimbra
{pgomes,camara,bento}@dei.uc.pt

Abstract

Complexity in software design is increasing rapidly, forcing development teams to be more efficient and more ingenious in their solutions. One of the fields that has been evolving is Software Reuse, which consists on using previous development knowledge in new projects. Due to the cognitive complexity, reusing software is a difficult task, especially when one spends more time in understanding and modifying old software, than building it from the scratch. This makes a great opportunity for tools that can help reusing software, and designing applications. In this paper, we propose analogical reasoning as part of such a tool. Analogical reasoning can produce innovative designs, or suggest new ideas to the designer, thus promoting creative solutions in the reuse of software.

Introduction

Software is becoming more complex and more prone to errors. A balance between software quality and functionalities provided by the software system (which are directly proportional to its complexity) is a constant concern of the software designers. This makes a great stress in the development process, rushing software companies in the creation of methodologies. Despite all the software methodologies developed in the last decades, software design is still more an art than an engineering field. Like architects, software designers frequently use their experience from the development of previous systems to design new ones. Most of the mature engineering fields make the reuse of components a rule of development, but in software engineering the reuse of components and/or ideas is not easy, given the conceptual complexity that software possesses. Thus, intelligent tools need to be at the disposal of software designers, in order to help them creating new systems using parts from previous ones, a matter that justified the investment on a dedicated research area: Software Reuse.

The main goal of software reuse is to shorten the development time of a software project improving its quality at the same time. This goal is very important to a software house, which has to produce complex software within a short time, and with profits. Software development has different levels of abstraction, each level dealing with several types of knowledge. To each of these levels corresponds a reuse level, ranging from code reuse (the most common form of software reuse), to project requirements reuse, passing by software design reuse. In the work presented in this paper we address the reuse of design knowledge.

Most of the tools and systems developed to reuse software (Basset 1987; Prieto-Diaz 1991; Katalagarianos and Vassiliou 1995; Fernández-Chamizo, González-Calero et al. 1996) only provide help in retrieving software entities, like classes, functions or specifications, from repositories. But reusing software involves also adaptation of the retrieved knowledge to the system being developed, which is usually left to the designer, since this is a more complex and demanding task. Analogical reasoning (Gentner 1983; Hall 1989; Holyoak and Thagard 1989) appeared as a technique that can be used to overcome some of the problems of the adaptation phase of software reuse, since it involves the transfer of ideas and solutions from other domains to the target domain. This not only provides a way to find solutions, but it also gives the opportunity to built new solutions and sometimes creative ones. Not only because they are novel and unexpected, but also because they can be better, simpler and more elegant. We believe analogical reasoning can achieve this on its own or with the collaboration of the software designer, providing the designer with ideas and alternatives that help to explore the solution space in a more efficient way.

We are developing a CASE (Computer Aided Software Engineering) tool, ReBuilder, which uses analogical reasoning to reuse software. Our goals are: to deploy this tool in a software development company that is our project partner; to help the development of software; and to motivate the production of creative solutions in the designers helped by ReBuilder. In our approach we represent software designs in UML (Rumbaugh, Jacobson et al. 1998) (Unified Modeling Language), a graphical language used to describe and document object oriented software designs, that is a standard for most of the software development companies.

In the next section, we present some issues related to Creative Design. Then, we show how analogy can help

software design reuse. Then we present our approach to analogical reasoning in software reuse using UML. We also describe an example of a simple problem. Finally, we draw some conclusions and present ongoing and future work on our system.

Creative Design

Design is the process of generating a structural description that complies with a set of functional specifications and constraints (Tong and Sriram 1992). Constraints can be structural, behavioral or resource limitations. The design product is a set of components and relations between them, making the design structure. The design process involves three main types of knowledge about the domain: functional, behavioral and structural. The mapping between these three types of knowledge is central to the design process (see Figure 1, (Reich 1991)).

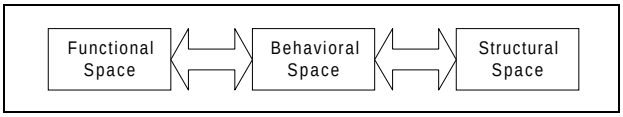

Figure 1 - The design process as a mapping process between functional, behavioral and structural spaces.

Design is classified as routine and non-routine (Tong and Sriram 1992; Gero and Maher 1993; Gero 1994b). Routine design is defined as a class of design where all the needed knowledge for the mapping process is available to the designer. Thus routine generated designs are instances of known class of designs. In non-routine design some knowledge for the mapping process is missing. When this lack of knowledge is located in the structural space, it originates a subclass of non-routine design known as innovative design. If there is knowledge missing in all the spaces it is called creative design. Innovative design creates new designs using values for the design variables, not commonly used in previous designs. The designs generated in creative design define new classes of artifacts, thus expanding the space of known designs.

In solving design problems, designers use old solutions. While in routine design old solutions are used to solve new situations with almost no change, non-routine design uses old designs in novel ways expanding the space of design solutions. We defend that Case-Based Reasoning (CBR) is a suitable framework for creative design, since it is a paradigm that uses old solutions to solve new problems (Kolodner 1993).

Kolodner and Wills (Kolodner and Wills 1993) claim that indexing cases accordingly to various perspectives is also important for case-based creative design. Kolodner and Wills propose the use of exploration processes that can search the case memory for cases that might be represented in a different way from the current problem representation. More recently Simina and Kolodner (Simina and Kolodner 1997) propose a computational model that accounts for opportunistic behavior, as a characteristic of creative behavior in case-based design.

Gero (Gero 1994a) defines five main creative design processes using design prototypes: combination, mutation, analogy, first principles and emergence. Combination involves the composition of two or more prototypes generating a new one. The mutation process involves a structural modification of one or more components of the design prototype. Analogy is defined as a mapping process between source and target design prototypes. First principles use knowledge about the design domain through the use of causal or qualitative models. Emergence is a process where additional properties of the design are identified, beyond the intentional ones.

Sycara & Navinchandra (Sycara and Navinchandra 1991; Sycara and Navinchandra 1993) produced another important work in this area. They propose a thematic abstraction hierarchy of influences as a retrieval method for case-based creative design. In this framework case organization provides the main mechanism for cross-contextual reminding which is very important in non-routine design.

Bhatta and Goel proposed an analogical theory of creativity in design (Bhatta and Goel 1997; Bhatta and Goel 1997). This theory is based on generic teleological mechanisms, which are behavior-function models. These models are abstracted from the case-specific structure-behavior-function models and are used for complex topological modifications in designs.

In ReBuilder we focus on the exploration of the space of old designs. We defend that exploration of space areas further away from the new problem area increases the probability of finding creative designs. Analogy can provide transference of knowledge between the new problem and areas of the search space far away from the new problem.

Creative Process and Creative Product

Creative design can be defined as a cognitive process that generates new classes of designs. During the design process the designer explores the space, discovering new pieces of knowledge enabling the creation of new design classes. The Creative Process is the process that generates new designs; its outcome is the Creative Product (see Figure 2).

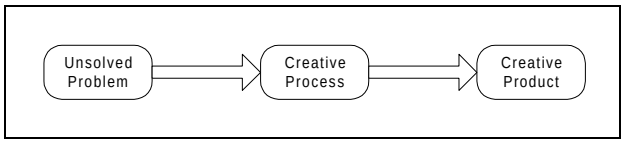

Figure 2 - Creative design path.

A cognitive process resulting into a product is considered creative if the product satisfies certain properties or attributes (Dasgupta 1994). These properties determine the product's creativity, thus defining guidelines to the

product's examiner. In the remaining of this section we present some of the main properties that characterize a creative design from two different perspectives: the creative process and the creative product.

The Creative Process

Processes for routine design are likely to be different from those for creative design (Gero and Maher 1993). This raises an important issue: does creative design involve intention from the point of view of the designer? This is a controversial question that the research community has not answered yet. For instance, can a certain degree of randomness be regarded as a characteristic of a creative process? Researchers working on evolutionist systems can say that it has its role in the process. But can chance be regarded in the same way? Can a person come up with a creative solution to a problem, even if there was no intention to solve the problem?

We will focus on more tangible aspects of the creative process having in mind that the processes for creative design are mainly different from those for routine design. Though we think that some of the reasoning processes applied in creative design are used in routine design.

One consequence of the definition of creative design is that possible solutions cannot be pre-established, at least explicitly. Otherwise it would be routine design. But the frontier between explicit and implicit definition of the possible solutions is blur. Possible solutions are defined by the knowledge available. If the space of solutions does not comprise all the possible solutions it must be incomplete and/or contradictory. These are two characteristics with which the creative processes must deal. Creative processes work with knowledge comprising distinct characteristics from those found in routine design processes. Creative processes must also comply with knowledge needed to perform cross-domain transfer of ideas, which is regarded as one of the types of reasoning associated to creative design (Sycara and Navinchandra 1991).

Design specifications comprise constraints and functional specifications. Both define the space of possible problems, thus constraining the space of possible designs. In creative design these spaces change during the problem-solving task. Two possible space modifications are addition and substitution (Gero 1994b). Addition consists in integrating new design specifications to the problem space or new designs to the solution space. In substitution, a region of a design space is substituted by another set of design specifications or solutions. Replacing design variables can perform this.

Reasoning processes modify the search space. During this modification the reasoning processes generate new design solutions. The generation of several design alternatives is important for exploration of the space of possible solutions and the space of possible problems, thus assisting the designer understanding the problem nature.

At this point two important concepts must be clarified, one is search and the other is exploration. Search is generally related with routine design and is defined as a computational process that requires the search space to be well defined (Gero 1994b). Exploration is a computational process that modifies the search space and is commonly associated with creative design. Though search is generally related to routine design, it can also be used in creative design. Flexible search mechanisms are needed for creative reasoning (Gero and Maher 1993). These mechanisms are responsible for the exploration of the design spaces.

Some of the main reasoning processes involved in creative design are: mutation, combination, analogy, reasoning from first principles and emergence (Gero 1994a). Mutation comprises the modification of a design structure in order to generate a new one. Another way for generation of new solutions is the combination of pieces from different designs. This process can work at different levels of the design - functional, behavioral or structural level. Analogy is regarded as one of the more important processes in creative design. It comprises mapping between a source design and a target design. It is a suitable mechanism for transfer of ideas across different domains, thus implementing cross-domain fertilization. Reasoning from first principles makes use of domain models in order to generate new designs. These models are causal or qualitative and can be viewed as generators from scratch of new regions in the search spaces. Emergence is a process in which additional design attributes are identified besides the intentional ones. This reasoning mechanism concerns to the ability to view things in new ways, which is a characteristic of creative reasoning (Partridge and Rowe 1994).

The Creative Product

There are two types of creativity: personal or psychological creativity, and historical creativity (Boden 1990; Dasgupta 1994). The former is related to the individual that evaluates the creative product and the later is associated with the evaluation by the social community. Both make use of domain knowledge, which is used to evaluate the product. While society knowledge comprises a set of conventional accepted rules or information, individual knowledge is more experience-based and specific to the individual.

The first thing that comes into mind when talking about creative design is novelty. This is a mandatory characteristic in a creative solution. The creative product must be something different from what the evaluator knows or thinks of. Evaluation of a creative product has to do with the confrontation of two sets of information. One is the information contained in the product and the other is the knowledge that the evaluator possesses or uses in the evaluation. If the information from the product does not make part of the evaluator knowledge set, then it can be said that the product is novel.

During the evaluation of the presumably creative product, the evaluator uses performance-measuring functions, in order to determine to which extent the product satisfies the problem requirements. These measuring functions make part of the evaluator’s body of

knowledge. If minimal performance thresholds are met, then the product is useful and solves the problem. This is another mandatory characteristic of creative solutions. The creative product must be appropriate and useful.

At a historical level, creative solutions must be novel in which regards to the society knowledge. In this way, evaluation of creative solutions comprises also a social aspect. This makes the automatic evaluation of creative products quite difficult and complex, leading to a two-step process. The first step is called internal validation and it consists on checking if the proposed design comprises all the intended requirements (checking for usefulness). This phase is also called verification and most times can be performed automatically. The second step evaluates the novelty of the design and is normally performed by the human in a computational system. It comprises the evaluation of the solution in regard to the information possessed by the examiner. This phase is called validation. What if the designer and the evaluator are the same entity? A benefit in the computational system doing the evaluation is the automation of the process, thus making it more efficient. Another advantage is the guidance that the evaluator can provide during the generation process. Nevertheless the computer system can hardly have the same sensibility to the problem as a human evaluator.

Analogy and Software Reuse

Creative processes work with knowledge comprising distinct characteristics from those found in routine design and also comply with knowledge needed to perform cross-domain transfer of ideas, which is regarded as one of the types of reasoning associated with creative design (Sycara and Navinchandra 1991). One of the main reasoning processes involved in creativity is Analogy (Gero 1994a). It comprises mapping between a source design and a target design, and it is a suitable mechanism for transfer of ideas across different domains, thus implementing cross-domain fertilization.

Analogical reasoning enables the exploration of new areas of the solution space, because it maps concepts from the target domain into a source domain. Since software reuse can be seen as a transfer of knowledge between a source project into a target one, analogy is a suitable mechanism to be used. Suggestions made by the system to the designer using analogical reasoning can also stimulate new ideas from the designer. It can also work in combination with the software designer, providing new ideas, while the designer evaluates and selects the suggested solutions.

The software design level is an appropriate level to work with because it deals with concepts, enabling the system to reason in a more abstract plan, one in which it can easily switch between different domains. But work at other levels of abstraction has also been developed. Maiden and Sutcliffe (Maiden and Sutcliffe 1992) proposed the reuse of software specifications through analogy. Working in a more abstract level than the software design level. The system works in requirement analysis, helping the system engineer to initially elicit the system's functionalities and constraints. They built a CASE tool that uses analogical reasoning to exploit specifications representing a wide range of applications held in a CASE repository. Cheng and Jeng (Jeng and Cheng 1993) use a different approach to analogy in software reuse. They use formal specifications to represent software components, and analogical reasoning to reuse the components. Having a formal specification of a software component makes easier the determination of its reusability and functionality. This approach has a major drawback, the software must be developed using the chosen formalism, which takes time and trained designers. Harandi and Bansali (Harandi and Bhansali 1998) describe a methodology for program derivation using analogy. Their work is at the code level, which restricts the use of analogical reasoning. They use their system to derive new Unix operating system environment programs, using heuristics to find good source analogues for the target problem. Our approach is different from the approaches of Maiden and Harandi at the reuse level, and from Cheng in the formalism used to represent software designs. The next section describes our approach and presents an example.

Analogical Reasoning with UML

UML is a graphical language, used to describe software systems. It comprises several diagrams capable of describing several aspects of a software system. These range from requirement analysis using Use Cases, to structural information using Class Diagrams, passing by behavioral knowledge specified using State Machines. ReBuilder uses UML as a representation formalism, providing the user with a common and worldwide acceptable software description language. ReBuilder uses several UML diagrams to specify and design a software system, usually class diagrams, which can be used for mapping concepts between different domains. An example of a class diagram is presented in Figure 3.

Designing a system's structure using UML involves two main steps. First creating the class diagram with the classes and interfaces needed to implement the system. This phase is called the conceptual design of the system's structure and involves only the main entities of the system represented as classes. In the second phase the designer needs to specify the class attributes and methods. This step requires from the designer a preview of which attributes and methods are necessary for the system implementation, which is not always easy. To assist the designer in building the class diagram we use analogical reasoning.

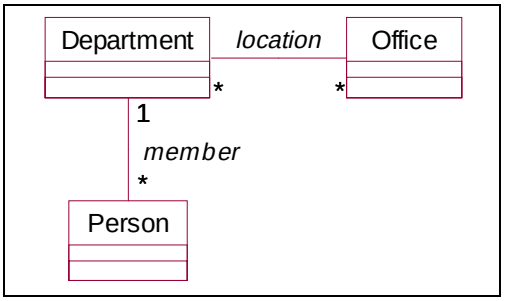

Figure 3 - A UML class diagram describing part of a system for a company management. Only classes are shown, because the diagram is presented at a conceptual level.

The analogical engine helps the designer providing mappings between the partial class diagram of the system being developed and class diagrams stored in a knowledge base of previous systems. Thus, it suggests new classes, attributes or methods to be added to the current class diagram. The process comprises four steps.

First, the analogical engine searches and identifies a set of candidate class diagrams from the knowledge base. To accomplish this task we use a structure-matching algorithm similar to SME (Falkeneheimer, Forbus et al. 1986). In order to guide the search we use an heuristic measure that ranks classes based on the class degree of importance taking into consideration the relations that the class possesses with other UML objects. Using this importance measure we can identify which classes should be used as probes in the search algorithm.

The second step comprises the ranking of the alternative mappings found by the structure-mapping algorithm. In this task we use the degree of matching between the current class diagram and the alternative diagram.

The third step is the presentation of the ranked alternatives to the designer so he/she can evaluate and select the most suitable one. In ReBuilder all the modifications to the current system design have to be approved by the designer.

The last phase consists in completing the current class diagram using the selected mapping. In this step new class, attributes and methods can be added to the class diagram.

Figure 5 shows a short example of the application of analogical reasoning. The UML class diagram presented in Figure 3 is used as the target for the analogical engine. The class diagram of Figure 4, was chosen from the UML case repository and is used to complete the target design, the resulting diagram is presented in Figure 5. As can be seen, the target diagram has now two additional classes: *Company* the analogous of *School* and *Client* the analogous of *Student*. Just to illustrate the idea, the *Department* class has gained some attributes, methods and relations, which came from its counterpart in the source diagram.

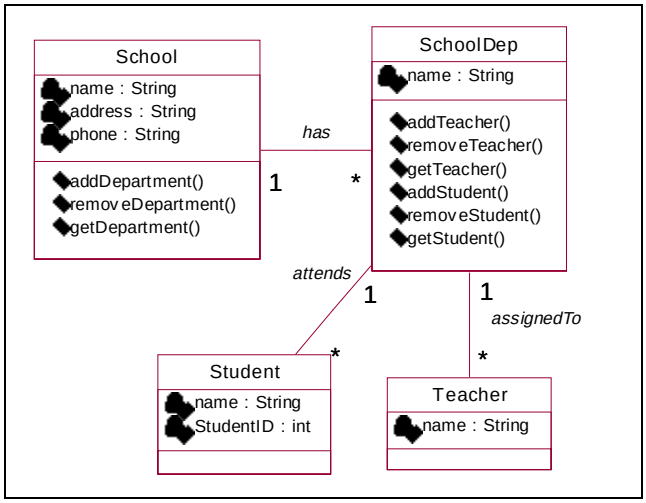

Figure 4 - The source UML class diagram used to complete the class diagram of figure 1.

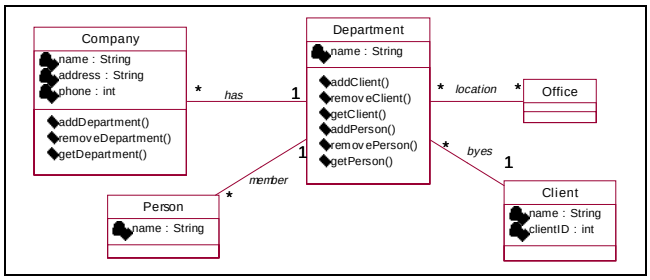

Figure 5 - The result of analogical reasoning between diagrams of figure 1 and 2.

Conclusion and Future Work

We think that presenting several alternatives to the designer helps the construction and completion of the current class diagram by exploration of the solution space. These alternatives can be found with analogical reasoning applied to the UML formalism. One of the main advantages of analogical reasoning is its capability to explore different domains, and to create new ideas from this exploration. Despite this major benefit, analogical reasoning has some limitations, such as the complexity and expensive computational work involved in the process, and also the creation of bizarre designs. Some of these problems can be solved using search-guiding heuristics, so that the search done by the analogical reasoning can be oriented to productive areas of the solution space.

At this stage we are finishing the implementation of the Knowledge Base and the basic retrieval mechanisms. So, the analogical reasoning module has not been implemented yet, but it is the next step in the process. We are also planning to explore in more detail the use of analogical reasoning in software reuse, especially at the specification level, which corresponds in UML to the Use Cases. Future

work on ReBuilder will also involve the use of Design Patterns as Case-Based Reasoning adaptation plans for software designs.

Acknowledgements

This work was partially supported by POSI - Programa Operacional Sociedade de Informação of Portuguese Fundação para a Ciência e Tecnologia and European Union FEDER, under contract POSI/33399/SRI/2000, by program PRAXIS XXI, and by Fundação Calouste Gulbenkian.

References

Basset, P. G. (1987). "Frame-Based Software Engineering." *IEEE Software*(July): 9-16.

Bhatta, S. and A. Goel (1997). *An Analogical Theory of Creativity in Design*. International Conference on Case-Based Reasoning (ICCBR 97), Providence - Rhode Island, USA, Springer-Verlag.

Bhatta, S. and A. Goel (1997). *Design Patterns; A Computational Theory of Analogical Design*. International Joint Conference on Artificial Intelligence (IJCAI'97).

Boden, M. (1990). *The Creative Mind: Myths and Mechanisms.* London, Weidenfeld and Nicolson.

Dasgupta, S. (1994). Creativity, Invention and the Computational Metaphor: Prolegomenon to a Case Study. *Artificial Intelligence and Creativity*. T. Dartnall, Kluwer Academic Publishers.

Falkeneheimer, B., K. D. Forbus, et al. (1986). *The structure mapping engine.* Sixth National Conference on Artificial Intelligence, Philadelphia, PA.

Fernández-Chamizo, C., P. González-Calero, et al. (1996). *Supporting Object Reuse through Case-Based Reasoning.* Third European Workshop on Case-Based Reasoning (EWCBR'96), Lausanne, Suisse, Springer-Verlag.

Gentner, D. (1983). "Structure Mapping: A Theoretical Framework for Analogy." *Cognitive Science* **7**(2): 155-170.

Gero, J. (1994). Computational Models of Creative Design Processes. *Artificial Intelligence and Creativity*. T. Dartnall, Kluwer Academic Publishers.

Gero, J. (1994). Introduction: Creativity and Design. *Artificial Intelligence and Creativity*. T. Dartnall, Kluwer Academic Publishers.

Gero, J. and M. L. Maher (1993). *Modelling Creativity and Knowledge-Based Creative Design*. Sydney, Lawrence Erlbaum Associates.

Hall, R. P. (1989). "Computational approaches to analogical reasoning; A comparative analysis." *Artificial Intelligence* **39**(1): 39-120.

Harandi, M. and S. Bhansali (1998). *Program Derivation Using Analogy*. Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA, Morgan Kaufmann Publishers, San Mateo, California.

Holyoak, K. J. and P. Thagard (1989). "Analogical Mapping by Constraint Satisfaction." *Conitive Science* **13**: 295-355.

Jeng, J.-J. and B. Cheng (1993). *Using Analogy and Formal Methods for Software Reuse*. IEEE 5th International Conference on Tools with AI.

Katalagarianos, P. and Y. Vassiliou (1995). "On the reuse of software: a case-based approach employing a repository." *Automated Software Engineering* **2**: 55-86.

Kolodner, J. (1993). *Case-Based Reasoning*, Morgan Kaufman.

Kolodner, J. and L. Wills (1993). *Case-Based Creative Design*. AAAI Spring Symposium on AI+Creativity, Stanford, CA, USA.

Maiden, N. and A. Sutcliffe (1992). "Exploiting Reusable Specifications Through Analogy." *Communications of the ACM* **35**(4): 55-64.

Partridge, D. and J. Rowe (1994). *Computers and Creativity*, Intellect Books.

Prieto-Diaz, R. (1991). "Implementing Faceted Classification for Software Reuse." *Communications of the ACM*(May).

Reich, Y. (1991). "Design Knowledge Acquisition: Task Analysis and a Partial Implementation." *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based Systems* **3**(3): 234-254.

Rumbaugh, J., I. Jacobson, et al. (1998). *The Unified Modeling Language Reference Manual*. Reading, MA, Addison-Wesley.

Simina, M. and J. Kolodner (1997). *Creative Design: Reasoning and Understanding*. International Conference on Case-Based Reasoning (ICCBR 97), Providence - Rhode Island, USA, Springer-Verlag.

Sycara, K. and D. Navinchandra (1991). *Influences: A Thematic Abstraction for Creative Use of Multiple Cases.* First European Workshop on Case-Based Reasoning.

Sycara, K. and D. Navinchandra (1993). *Case Representation and Indexing for Innovative Design Reuse*. Workshop of the 13th International Joint Conference on Artificial Intelligence, France.

Tong, C. and D. Sriram (1992). *Artificial Intelligence in Engineering Design*, Academic Press.